Parallelization, Distribution and Scaling of Multi-Simulations on Multi-Core Clusters, with DACCOSIM Environment

Cherifa Dad, Stéphane Vialle, Mathieu Caujolle, Jean-Philippe Tavella, Michel Ianotto

Technical Report - v3

Parallelization, Distribution and Scaling of Multi-Simulations on Multi-Core Clusters, with DACCOSIM Environment

Cherifa Dad[†], Stephane Vialle[†], Mathieu Caujolle[‡],
Jean-Philippe Tavella[‡], Michel Ianotto[†]

[†] UMI 2958 GeorgiaTech-CNRS, CentraleSupelec,
University Paris-Saclay, 57070 Metz, France
firstname.lastname@centralesupelec.fr

[‡] EDF Lab Paris-Saclay, 91120 Palaiseau, France
firstname.lastname@edf.fr

March 23, 2016

Abstract

DACCOSIM is a multi-simulation environment for continuous time systems, relying on FMI-CS standard and making easy the design of a multi-simulation graph. Moreover, DACCOSIM has been specially developed to run on multi-core PC clusters, in order to achieve speedup and size up on large scale multi-simulations. However, the distribution of the multi-simulation graph remains complex and is still under responsibility of the simulation developer.

This paper introduces DACCOSIM parallel and distributed architecture, and our strategies to achieve efficient distribution of multi-simulation graph on multi-core clusters. Some performance experiments on two clusters, running up to 81 active components (FMU) and using up to 16 multi-core computing nodes, are shown. Performances measured on our faster cluster exhibit a good scalability, but some limitations of current DACCOSIM implementation are discussed.

Contents

1 Introduction

Complex systems are characterized by the interconnection of numerous and heterogeneous cyber components (e.g. controllers) and physical components (e.g. power grids). For EDF (the major French utility company), the smart power grids will extensively rely on new control functions (i) to increase the grid efficiency, reliability, and safety, (ii) to enable better integration of new assets (e.g. distributed generation and alternative energy sources), (iii) to support market dynamics and manage new interactions between established and new energy players. *Cyber-Physical Systems* (CPS) design involves multiple teams working simultaneously on different aspects of the system. Especially, the development of a smarter electrical grid requires to reuse models and tools often based on separate areas of expertise.

This heterogeneity led EDF to investigate coupling standards such as the *Functional Mock-up Interface* (FMI[1]) initiated by Daimler AG within the ITEA2 MODELISAR project and now maintained by the *Modelica Association*[2]. More precisely, EDF chose its FMI-CS (FMI for co-simulation) part because this operation mode allows to export models as active components called FMUs (Functional Mock-up Units), each FMU being a self-contained archive file including a model and a numerical solver. As an additional benefit, the model IP is readily protected when models are exported as FMUs from FMI-CS compliant modelling tools.

For EDF, it is vital to develop agile modelling and simulation in order to design and validate distributed operating functions a long time before performing tests on experimental sites. The *Distributed Architecture for Controlled CO-SIMulation* (DACCOSIM software [8]) developed by EDF and CentraleSupelec is a part of the answer to this problematic. It is aimed at simulating large and complex multi-physics systems on multi-core PC clusters (the standard scalable computing architecture), despite some load unbalance and important communications inherent in our multi-simulations. This paper introduces the parallel and distributed architecture of DACCOSIM, and some scaling experiments on PC clusters (running up to 81 FMUs and using up to 16 multi-core cluster nodes). Finally we list some issues and future works to achieve better speedup and size up.

These researches are carried out by the RISEGrid[3] institute, founded by EDF and CentraleSupelec with the ultimate goal to design the smart electric grids of the future.

2 Context and related works

A FMI based multi-simulation is defined by a FMU graph, where all inputs and outputs need to be correctly initialized during a mainly iterative phase called co-initialization. EDF has achieved the co-initialization phase of DACCOSIM reusing

[1] https://www.fmi-standard.org/downloads

[2] https://www.modelica.org/

[3] http://www.supelec.fr/342_p_36889/risegrid.html

its expertise in the famous Newton-Raphson algorithm (very popular for power flow calculations) in conjunction with recent works done at UC Berkeley on dependency cycles analysis [2] (which are now possible with the latest version 2.0 of the FMI standard). The global dependencies graph automatically deduced from the FMU graph is valuable to solve algebraic loops with a parallel Newton-Raphson algorithm staged at the initialization mode of the multi-simulation. Then, the FMU graph enters a loop of parallel multi-simulation time steps, concurrently running each FMU at each time step, without the need for additional power flow calculations between two consecutive steps. From this point of view, the current version 2.0 of the FMI standard seems sufficient for EDF as its current use cases do not report non convergence examples due to algebraic loops in the step mode.

So, our problematic is to distribute a FMU graph on a multi-core PC cluster. This looks like a task graph distribution problem, which is an important research field. For example, [5] proposes a scheduling algorithm on heterogeneous distributed architectures for static tasks modeled by Directed Acyclic Graph (DAG), and [6] introduces mapping strategies of hierarchically structured multiprocessor tasks on multi-core clusters, which is our target architecture. However, our DACCOSIM FMU graph is not hierarchically structured, and is a DAG with very few task dependencies. All FMU tasks can run in parallel when starting a new time step, and when all computations are finished all inter-FMU communications can start and occur in parallel [8]. Then, when all communications are achieved, all FMUs can enter a new time step. There are no dependencies between our FMU task computations, and usual solutions of task graph distribution are not really adapted to our problem. The basic version of our FMU graph working is closer a classical *Bulk Synchronous Parallel* (BSP) model [12], which is well known in parallel computing, but our FMU tasks are heterogeneous and load unbalanced. Moreover, a computationally big FMU can not be split into smaller ones as it would require to design new mathematical models. So, our problem is not a classical *Simple Program Multiple Data* (SPMD) scheme following a BSP model. Our FMU graph has heterogeneity and load unbalance typical of generic task graphs. It is also possible to consider our FMU graph like a kind of time stepped *Multiple Program Multiple Data* application running on a computing cluster and aiming to reach high performances [3]. Identification of an efficient distribution of our FMU graph thus requires to design a specific algorithmic solution.

Some others multi-simulation environments interconnect some continuous time based simulators, like EPOCHS [10] and INSPIRE [9], or even some FMUs, like C2WT [1], through a HLA logical event bus [11]. This bus relies on a Run-Time Infrastructure (RTI), ensuring event routage and right synchronization between simulators. Many RTI implementations are distributed, and can activate concurrently simulators on different computing nodes. However, parallelism of the system depends on the abundance of *safe* events concentrated in a *lookahead* time interval [7, 13], and our time stepped FMU graph contains more potential parallelism.

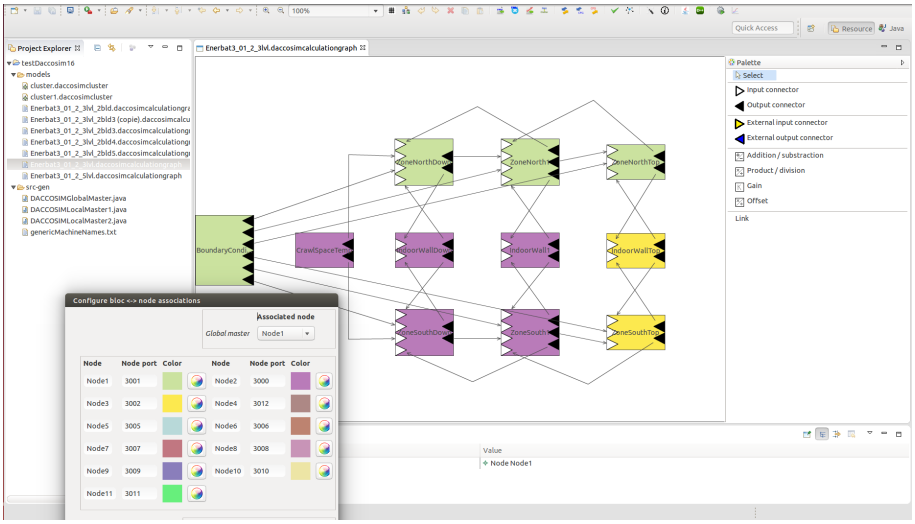

Figure 1: DACCOSIM GUI: FMU graph definition and configuration

3 DACCOSIM environment

3.1 DACCOSIM software suite

DACCOSIM has been designed to achieve multi-simulations of continuous time systems, discretized with *time steps*, and running solvers using constant or variable time steps (to maintain the requested accuracy with the minimal amount of computations, whatever the dynamic of the system). It consists in two complementary parts: a *Graphic User-friendly Interface* (GUI) and a *dedicated computation package*.

The GUI developed in Java facilitates the complex systems studies by designing the multi-simulation graph (Figure 1), i.e. the FMUs involved and the variables exchanged in-between, defining the resources used by the simulation (local machine or cluster), configuring the simulation case (duration, co-initialization method, time step control strategy...) and implementing the graph into DACCOSIM master tasks managing the simulation.

The *dedicated computation package* controls all task execution issues relative to the multi-simulation: co-initialization, local or distributed computation steps, fixed or variable time step control strategies, detection of state events generated inside FMUs, inter-FMU communications, distributed and hierarchical decision process... The Java version of DACCOSIM relies on JavaFMI[4] and is available for both Windows and Linux operating systems, whether 32-bit or 64-bit.

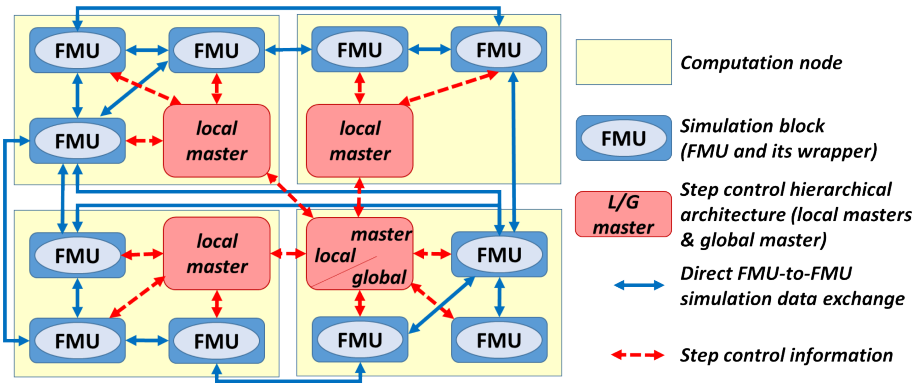

Figure 2: Distributed DACCOSIM architecture, with a hierarchical *master*

3.2 Parallel and distributed runtime architecture

The *dedicated computation package* of DACCOSIM includes a *parallel and distributed runtime architecture*, designed to take maximal advantage of any cluster of multi-core nodes. A DACCOSIM simulation executes a series of time steps composed of three stages: the time step computations of the FMUs (independent computations), the communication of the FMU outputs to the connected inputs (many small communications), and the simulation control by the hierarchical control master (information gathering, next operations decision and order broadcasting). All these operations include potential parallelism exploited by DACCOSIM runtime.

As inside one computing step all FMUs can achieve their computation concurrently, DACCOSIM architecture distributes FMUs on cluster nodes, encapsulates FMUs of a same node with different threads and implements a *hierarchical (and distributed) control master* to manage all threads and FMU operations. Figure 2 illustrates this architecture. The *hierarchical control master* is composed of a unique global master located on one cluster node, in charge of aggregating the control data coming from the local masters located on the different nodes, and taking decisions based on these information. The global master also assumes the role of the local master on its node. Every local master aggregates control data from the FMUs on its node, and sends synthesized control information to the global master. All master tasks run concurrently.

DACCOSIM uses FMUs in *co-simulation mode* (*FMI-CS*, see 1): they embed their solvers and are implemented as dynamic libraries (enhanced with meta-data XML files). A *FMU wrapper* thread encapsulates each FMU, calls its computation function to achieve a time step progress, and sends each of its outputs to the connected inputs. Three others threads are associated to each FMU, as illustrated on Figure 3: one receipts the output values coming from other computing nodes (cur-

[4]SIANI, University of Las Palmas, Spain: JavaFMI (2016), https://bitbucket.org/siani/javafmi/wiki/Home

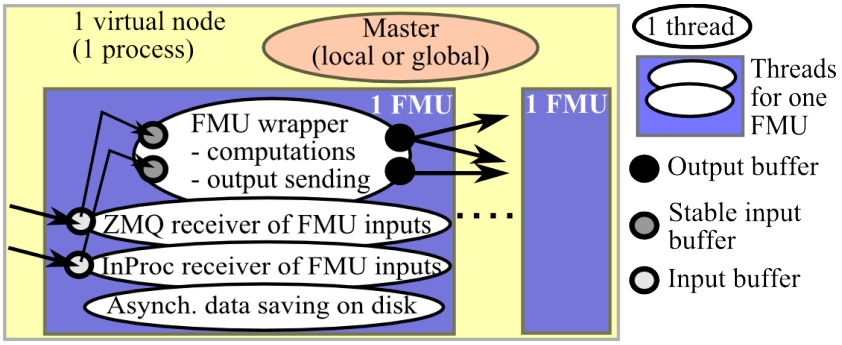

Figure 3: Multithreaded implementation of a virtual node

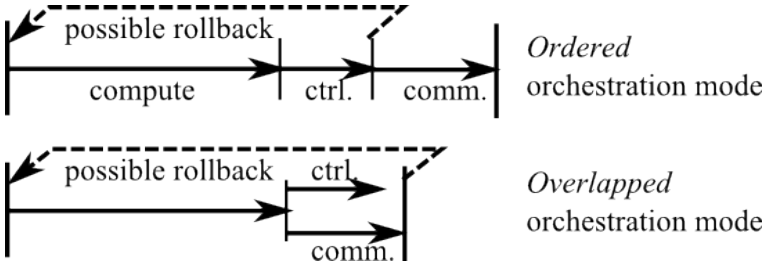

Figure 4: Two orchestration modes of a multi-simulation step

rently across ZeroMQ[5] middleware), one achieves similar receipt but from FMUs located on the same node (not going through the middleware to run faster), and one stores simulation results on disk (asynchronously and per block). Received input values are stored into buffers and distributed to the real FMU inputs before the start of each FMU computation step, so that the inputs remain stable during the computations.

When using variable time steps, the control master gathers and analyses some time step results. It decides if they are valid and if the simulation can enter the next step with the same or a larger time step, or if the simulation has to roll back and continue with a smaller time step (to reach the required accuracy). Two *orchestration modes* are available in DACCOSIM (see Figure 4). The *ordered* mode executes the three phases of each time step in order: (1) FMU computations, (2) communication and control with the hierarchical master, and (3) inter-FMU communications if the master has validated the time step results. The *overlapped* mode overlaps the inter-FMU communications with the control master operations (phases 2 and 3). If the control master requires a rollback, the received input values are forgotten and the FMU states at the beginning of the time step are restored. But when the dynamic of the simulated system is limited (no turbulence), there are few rollbacks and the *overlapped* mode reduces the simulation time. When the dynamic is high, many rollbacks can appear and a lot of inter-FMU communication phases can be

[5]http://zeromq.org/

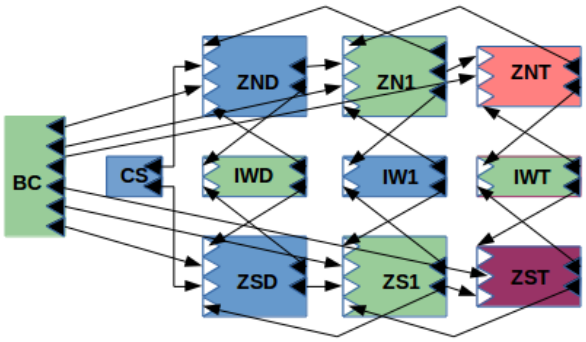

Figure 5: Best distribution (4 nodes) of *cl5* benchmark with 1 building

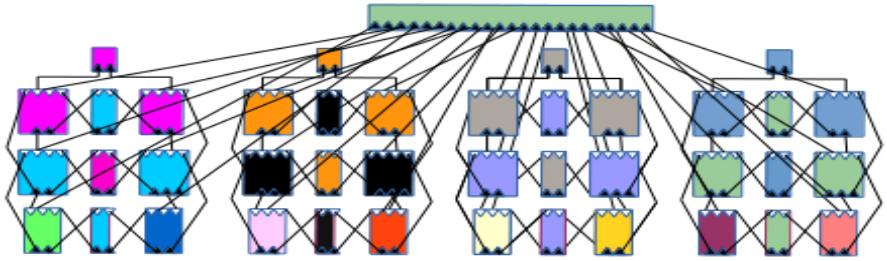

Figure 6: Scaling on 16 nodes of heat transfer *cl5* benchmark with 4 buildings

achieved needlessly. Currently the orchestration mode has to be set by the user when running the multi-simulation. In the future it could switch dynamically and automatically depending on the frequency of the appearing rollbacks.

A deployment tool (*dacrun*) completes the DACCOSIM software suite, and makes easy the deployment of a multi-simulation on a set of cluster nodes. It is compliant with the *OAR*[6] environment, allowing to allocate nodes and run interactive or batch jobs on a PC cluster, but can be easily adapted to any similar cluster management environment. However, design of a multi-simulation graph distribution on different cluster nodes is the responsibility of the multi-simulation developer. An automatic distribution of the FMUs is under investigation, according to the lessons learnt from our experiments (see next sections).

4 Testbeds description

This section introduces our scalable benchmark applications and technical features of our two PC clusters.

[6]https://oar.imag.fr/

	Walls	Zones
Complexity level 1	5	482
Complexity level 2	10	582
Complexity level 3	25	882
Complexity level 4	50	1382
Complexity level 5	500	10382

Table 1: Number of state variables function of the complexity level

4.1 Benchmark application

Our test case is a simplified industrial case provided by EDF R&D, representing heat transfers in a set of n three-floor buildings, with two zones per floor separated by an indoor wall. The entire multi-simulation includes $1 + 10 \times n$ FMUs, interconnected as illustrated in Figures 5 and 6. The unique FMU *BC* is in charge of the thermal boundary conditions (e.g. actual temperatures recorded in a French suburb) which are supposed identical for all the floors in every building. A complementary FMU *CS* models the crawl space temperature for the lower floor of a building, and each building having a specific FMU *CS*. The FMUs *ZNx* (resp. *ZSx*) represent the Northern (resp. Southern) part of every floor, each being designed with Modelica differential equations modeling physical phenomena such as conduction, convection and solar radiation. The FMUs *IWx* detail the behavior of the wall between the floor zones depending on different insulating properties.

All these FMUs are equation-based only, and modeled by encapsulated arrays of records with changeable size to propose 5 levels of complexity (size/weight) for each FMU (see table 1). No control or temperature regulation is considered in this benchmark, the data exchanged between these FMUs are temperatures and thermal flows. We designed two benchmarks running $1 + 10 \times n$ FMUs.

The *cl5* benchmark includes high complexity level FMUs, and exhibits negligible communication times compared to the computation ones. The *cl3* benchmark includes medium complexity level FMUs and has significant communications.

4.2 Experimental PC clusters

Two PC clusters of CentraleSupelec have been used for our experiments. The first one has a 10 Gigabit/s Ethernet interconnect, and each node includes one Intel Sandy Bridge processor with 6 hyperthreaded physical cores (6 Cores/12 Threads) at 3.2 GHz, and 8 GByte of RAM. The second one has a 1 Gigabit/s Ethernet interconnect, and each node includes one Intel Nehalem processor with 4 hyperthreaded physical cores (4 Cores/8 Threads) at 2.6 GHz, and 6 GByte of RAM.

Sandy Bridge cluster - 10Gbit/s - 1x Sandy Bridge 6C/12T											
FMU Id	ZNT	ZST	ZN1	ZND	ZS1	ZSD	IWD	IWT	IW1	BC	CS
$T_{calc}(s)$	**52.5**	**51.9**	**28.1**	**28.0**	**27.9**	**27.8**	0.04	0.04	0.03	0.02	0.01
$T_{ctrl+sync}(s)$	0.14	0.61	24.8	24.9	25.1	25.1	53.9	53.9	53.9	54.0	54.1
$T_{comm}(s)$	0.11	0.17	0.10	0.12	0.11	0.11	0.11	0.11	0.07	0.08	0.03

Table 2: Execution of *cl5* benchmark with one FMU per node

5 Parallelization and distribution methodology

To efficiently distribute our multi-simulations on a cluster of multi-core nodes, our approach has consisted in (1) pointing out the technical locks through experiments of a realistic use case, and (2) in designing a methodology to improve our experimental performances and to identify some efficient *distribution patterns* of elementary multi-simulations. Then, we will reuse these distribution patterns to quickly build larger scale multi-simulations and to conduct scaling experiments (see section 6).

5.1 FMU characterizations

FMUs are seen as black boxes during a DACCOSIM execution. However, we need to evaluate their computation times, inter-FMU communication times, and control times (communications with their master and re-synchronization) in order to establish the best deployment of k FMUs on n multi-core nodes. Moreover, to provide a realistic assessment, we measure these execution times when FMUs are connected and submitted to real input data at each time step, since the computations might change otherwise. So, we deployed the FMU graph of our *cl5* benchmark on our PC cluster, setting only one FMU per node so that they do not disturb each other. Of course, we use the *ordered* orchestration mode of DACCOSIM, not overlapping the control and inter-FMU communication steps, in order to measure accurately this communication time.

The T_{calc} line of Table 2 shows the computation time consumed by each FMU during these multi-simulation trials: the benchmark appears to run 2 big, 4 medium and 5 very small tasks. Almost each of these FMUs implements an *energy model* of a building zone, function of its size, location in the building, building materials... To split a big FMU into two medium ones would require to redesign some models instead of reusing already existing and validated components. So, a smart load balancing strategy is required to efficiently run this multi-simulation on a PC cluster without any change.

The $T_{ctrl+sync}$ time seems as significant as the T_{calc} time ($54.1s$ on the smallest FMU). But the global master resynchronize all FMUs at the end of each *control and synchronize* sub-step, making the small FMUs spend their time waiting for the big ones. In fact, the *control* part of $T_{ctrl+sync}$ is petty in this benchmark with constant time steps.

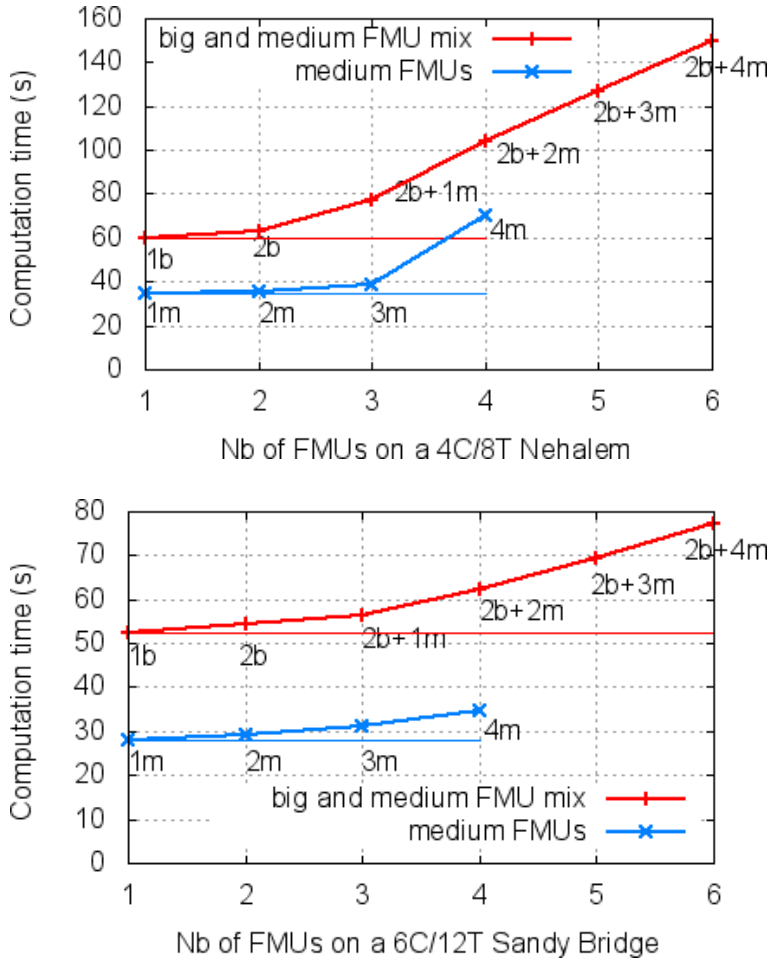

Figure 7: One node comput. times of *cl5* benchmark for different mix of FMUs

Finally, T_{comm} line of Table 2 shows the communications of this first benchmark are negligible. But we will see at section 5.4 this is not true for all benchmarks.

5.2 Multi-core node experiments

Our final objective remains to get a maximal performance on our clusters of multi-core nodes, achieving an optimal distribution of our FMUs on the cluster nodes. To reach this goal we have to take into account the very different computing loads of our FMUs, as characterized at section 5.1, but also to study the total computing time of a subset of FMUs and multithreaded FMU wrappers sharing a multi-core node. So, we have run from one to four medium FMUs on one node of our clusters, and then one big, two big and up to two big and four medium FMUs on one node to study their parallelization efficiency on the different cores. Figure 7 summarizes

nb of nodes	FMU distribution on cluster					
1	2 big (b) + 4 medium (m) + 5 small (s)					
2	2b	4m + 5s				
3	1b	1b	4m + 5s			
4	**1b**	**1b**	**2m + 2s**	**2m + 3s**		
6	1b	1b	1m + 1s	1m + 1s	1m + 1s	1m + 2s

Table 3: Load balanced FMU distributions of a 1-building *cl5* benchmark.

the computation times of our *cl5* benchmark elapsed on one node of our Nehalem cluster (top) and on one node of our Sandy Bridge cluster (bottom). We observe the parallelization on the different cores is not perfect. Horizontal lines are the ideal computation time when running up to 4 medium load FMUs on 4 physical cores (it is the execution time of the longer medium FMU). But the thick curves show the computation time increases significantly when running concurrently four medium load FMUs on a 4C/8T Nehalem processor, and increases regularly on a 6C/12T Sandy Bridge processor. When running 2 big FMUs and adding 1 to 4 medium ones, the horizontal lines show the expected execution times up to 4 FMUs on a 4C/8T Nehalem, and up to 6 FMUs on a 6C/12T Sandy Bridge. But the thick curves exhibit serious increase of the computation times, specially when running more FMUs than the number of physical cores on the Nehalem.

It appears we can run only 2 or 3 FMUs (with their associated threads) on a 4C/8T Nehalem, and 3 or 4 FMUs on a 6C/12T Sandy Bridge, to achieve a good parallelization with limited computation time overhead. Considering only the computing load, ideal solution seems to run a number of FMUs close to half of the number of physical cores.

5.3 Multi-core PC cluster experiments

Our heat transfer *cl5* benchmark includes only 11 FMUs, and 5 have very short computation times. So, deployment optimization consists in efficiently distributing 6 FMUs (2 big and 4 medium) on a set of multi-core nodes, taking into account the parallelization of several FMUs on one node is imperfect. Considering the multithreaded execution times measured at section 5.2, we computed the FMU distributions ensuring better load balancing from 2 up to 6 nodes (see table 3) and conducted experiments on our clusters.

Figure 8 bottom shows the times measured on our Sandy Bridge cluster are close to the expected ones. All nodes synchronize at each simulation step, and inter-FMU communication and control operations are negligible in this benchmark (see section 5.1). So, we computed the *planned computation time* of the multi-simulation as the longest computation time of all nodes, and we estimated the computation time of one node from the measures introduced at section 5.2. The experimental maximum computation time (among all nodes) appears to be close to the planned computation time on Figure 8 bottom, and the total execution time is

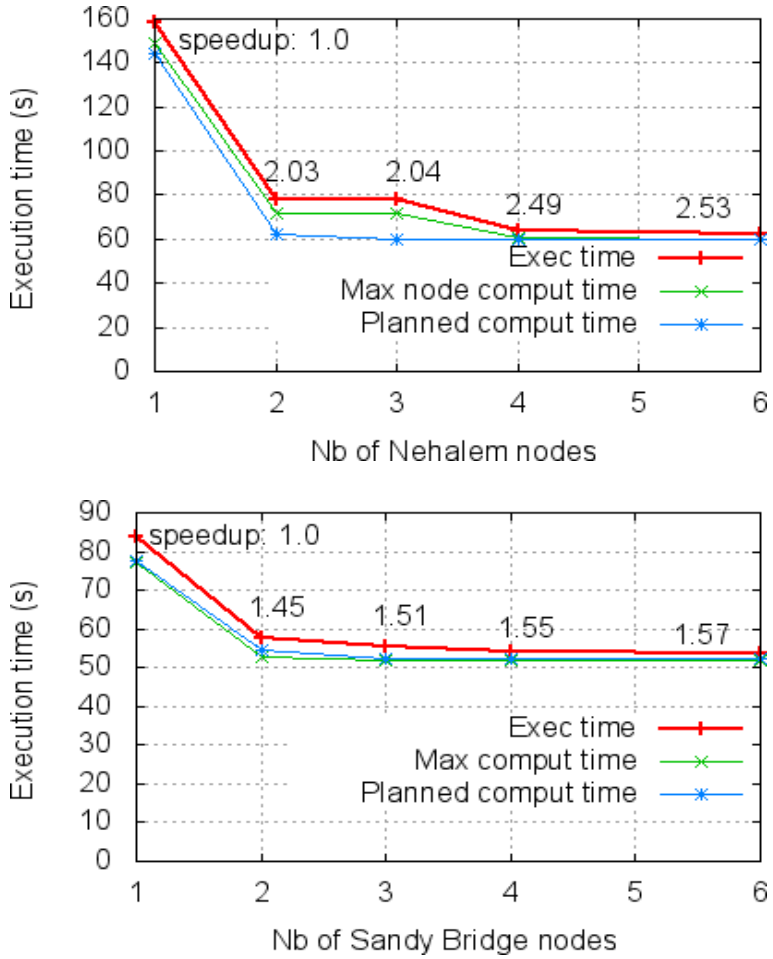

Figure 8: Execution and computation times of *cl5* benchmark on clusters.

a little bit longer due to the execution of the remaining 5 small FMUs and to the communication and control times. So we can allocate only 2 nodes and achieve a speedup of 1.45, or up to 6 nodes and achieve a speedup close to 1.57 (beyond 6 nodes the speedup remains unchanged). Using 3 or 4 nodes and the associated distribution described in table 3 seem the better compromise for this 1-building *cl5* benchmark running on our Sandy Bridge cluster.

On our Nehalem cluster (Figure 8 top), there are differences between the *planned computation time* and the measured one on 2 and 3 node configurations. Running the 5 remaining small FMUs on these old 4C/8T nodes seems to disturb the computations. We saw at section 5.2 these Nehalem nodes are more sensitive to concurrent executions of FMUs than the Sandy Bridge nodes. Using 4 nodes and the associated distribution appears to be the best compromise for our 1-building *cl5* benchmark running on our Nehalem cluster.

Finally, we consider the distribution on 4 nodes introduced in table 3 is the

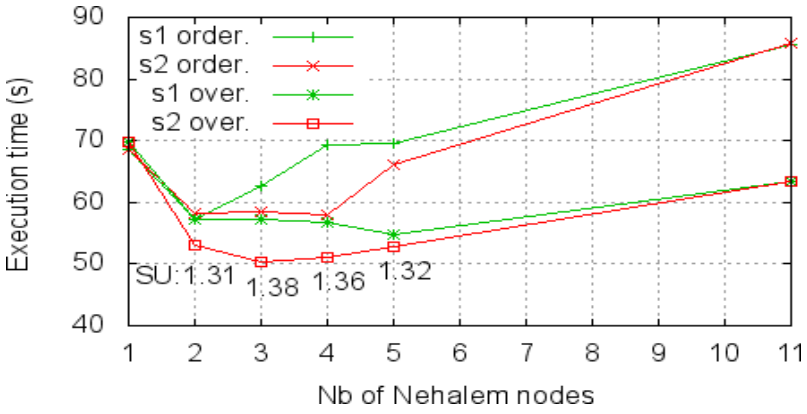

Figure 9: Exec. time of *cl3* benchmark on our Nehalem cluster

strategy 1: "load"	
load:	26.0s; 26.0s; 23.0s
nb intra-node comm.:	14

strategy 2: "load+comm"	
load:	29.8s; 26.0s; 24.0s
nb intra-node comm.:	16

Table 4: Load balancing and number of intra-node communications on 3 nodes function of the FMU distribution strategy

right *distribution pattern* for this 1-building *cl5* benchmark.

5.4 Multi-criteria distribution strategy on PC clusters

Our *cl3* benchmark implements the same multi-simulation graph as the *cl5* one, but with less computations. So, communications become more frequent and *cl3* benchmark is harder to speedup. We followed the 3-step methodology established in previous sections:

1. We characterize the load of the different FMUs, running one FMU per node. We identified 4 kinds of FMUs: 2 big ones, 4 medium ones (75% of the big FMU load), 3 small ones (17% of the big FMU load) and 2 FMUs with insignificant loads. Moreover, we observed the inter-FMU communication time could reach half of the computation time.

2. We measured the real computation time on one node running several FMUs, and parallelization appeared imperfect again.

3. Considering the real computation times of different sets of FMUs per node, we computed and experimented different distributions of our multi-simulation graph. We achieved our experiments on our Nehalem cluster, that does not efficiently parallelize more than 2 or 3 FMUs per node, and seemed our best testbed to evaluate how our (only) 11-FMU benchmark could scale.

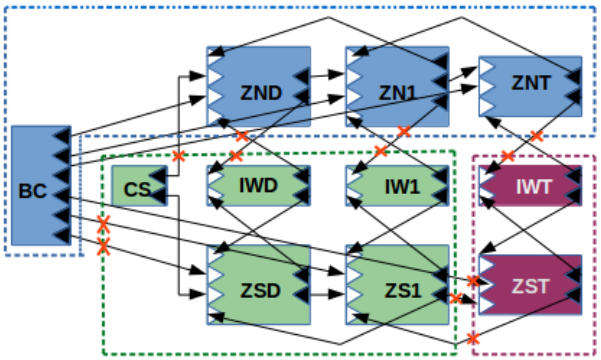

Figure 10: Best identified FMU distribution for 1-building *cl3* benchmark (3 nodes have to be used)

Figure 9 shows the execution time achieved on our Nehalem cluster by our *cl3* benchmark, function of the orchestration mode (*ordered* or *overlapped*) and the distribution strategy. *s1* strategy focuses on the load balancing of the FMUs across the cluster, while *s2* strategy tracks a compromise between the load balancing and the maximization of the number of *intra-node* communications (the inter-FMU communications achieved inside a same node).

When using three nodes, table 4 shows *s2* strategy sacrifices a little bit the load balancing of FMU computations to increase the intra-node FMU communications. Finally, the *s2* strategy reaches better performance and the *overlapped* orchestration mode leads to significantly decrease the execution time. Both they allow to achieve a speedup close to 1.38 on three nodes for this *cl3* benchmark with heterogeneous computation tasks and important communications. Figure 10 shows the best FMU distribution, achieved on three nodes, which has become the *distribution pattern* of this 1-building *cl3* benchmark on our Nehalem cluster.

6 First scaling approach and experiments

All experiments introduced in this section use the *overlapped* orchestration mode, and avoid to go through the middleware for internal node communications. This configuration appears the most efficient on our benchmarks and clusters. We measured multi-simulation times with result storage (approximately $850MB$ per run), however we stored these results on the local disks of the computing nodes, in order to avoid fluctuations due to network traffic on our LAN.

6.1 Size up achievement based on distribution pattern replication

We conducted many experiments to measure *size up* of our multi-simulation benchmarks: we increased both the problem size (number of buildings simulated) and the number of computing ressources (number of cluster nodes), attempting to keep

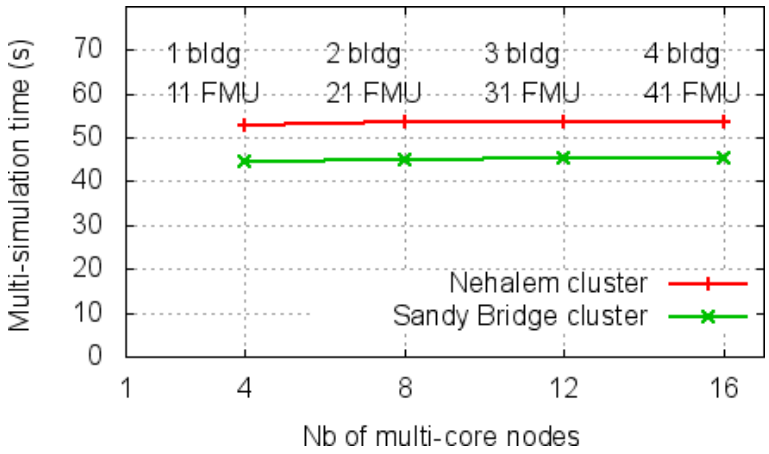

Figure 11: Size up experiments on *cl5* benchmark

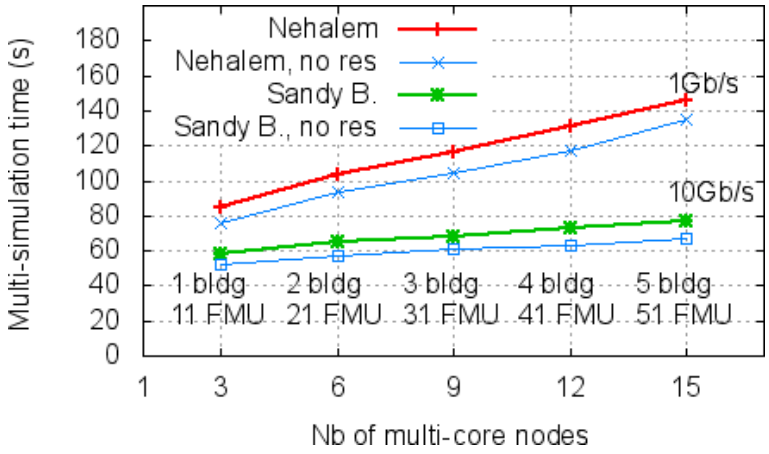

Figure 12: Size up experiments on *cl3* benchmark

constant the execution time. When processing larger multi-simulations we replicate the distribution pattern identified for one building (see figure 5 and 6). So we attempt to obtain:

$$T(b\ buildings, b \times n_1\ nodes) = T(1, 1 \times n_1)$$

With n_1: the optimal number of nodes to simulate 1 building

Our previous experiments has shown the optimal distribution for one *cl5* building requires 4 nodes, so we simulated 2, 3 and 4 buildings on 2×4, 3×4 and 4×4 nodes. Figure 11 shows this approach has fully succeeded for the *cl5* benchmark. We obtained approximately the same execution time: $53s$ on our Nehalem cluster with an $Eth\ 1Gb/s$ interconnect and $45s$ on our Sandy Bridge cluster with an $Eth\ 10Gb/s$. These results were expected, as the *cl5* benchmark includes mainly computations and our replication strategy of distribution pattern avoids the building

simulations disturb each others.

At the opposite, the *cl3* benchmark includes significant amount of communications compared to the amount of computations. We replicated our 3 nodes distribution pattern, and we simulated 2, 3, 4 and 5 buildings on 2×3, 3×3, 4×3 and 5×3 nodes. Figure 12 shows we obtained unperfect but interesting performances on our $10Gb/s$ cluster (bottom thick curve). Multi-simulation times increase by $+31\%$: from $58s$ up to $76s$ when processing 1 building on 3 nodes up to 5 buildings on 15 nodes. On our $1Gb/s$ cluster the times increase by $+72\%$ (top thick curve). This experiment shows the impact of the cluster network bandwidth, but we also observed the FMUs exchange mainly small messages. So, a high bandwidth and a low latency seem both necessary to achieve good size up of DACCOSIM simulations.

Thin curves on figure 12 show the execution time when the multi-simulation results are not stored. No result storage means no asynchronous IO thread running in parallel with all others on each node, and no time spent writing on disks. On *cl3* benchmark it leads to smaller execution times but size up benchmarks still exhibit regular increase of execution time. So, we can probably improve our result storage mechanism, but it is not the main limitation of our size up benchmark.

6.2 Speedup evolution when increasing multi-simulation size

Finally, figures 13 and 14 exhibit multi-simulation times of *cl3* benchmark for 4, 5 and 8 buildings, when increasing the number of computing nodes. We replicated our distribution pattern (1 building on 3 nodes for *cl3* benchmark), to deploy for example 4 buildings on $12 = 4 \times 3$ nodes. Then we grouped 2 buildings per 3-node block to deploy 4 buildings on 6 nodes, and finally we grouped the 4 buildings on 3 nodes. So, all our deployments are based on replication of the previously identified optimal deployment of 1 building. This straightforward approach does not track a global optimized deployment, but allows to quickly configure large scale experiments.

Logarithmic scales on both axis lead to expect straight lines with -1 slope for ideal distributions, corresponding to theoretical times $T(n\ nodes) = T(1\ node)/n$. Experimental time curves roughly look like straight lines, but the slope appears stronger on 10Gb/s cluster, showing again the impact of the interconnect performance on our distributed multi-simulations. When simulating 4 or 5 buildings (middle and bottom curves), it was bearable to run and measure execution times on 1 multi-core node and to compute speedup. On 10Gb/s cluster we achieved significant speedup close to 7 on 12 nodes and to 9.7 on 15 nodes (see figure 14), compared to multi-threaded executions on one node. When simulating 8 buildings we did not waste time to run long computations on one node: large problems are not intended for mono-node executions (and perhaps could not fit into one node memory). But we observe the 8-building curves are similar to the 4-building ones with better decrease. So, DACCOSIM can efficiently run larger problems on greater number of nodes and *scale* our multi-simulations (with a fast cluster network).

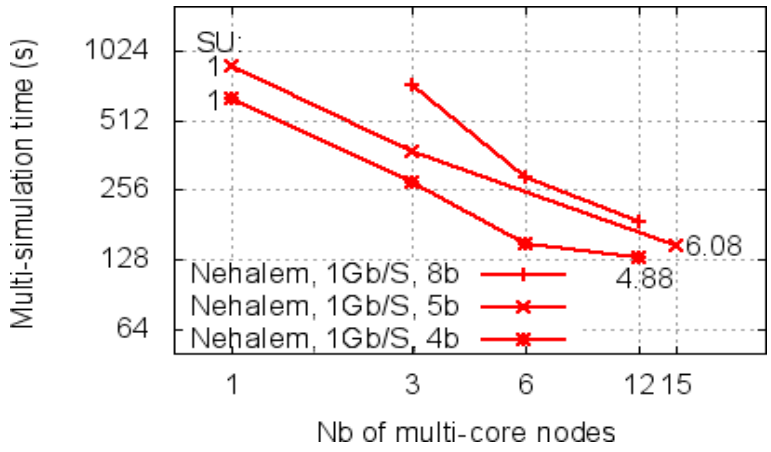

Figure 13: *cl3* benchmark time with 4, 5 and 8 buildings on 1Gb/s cluster

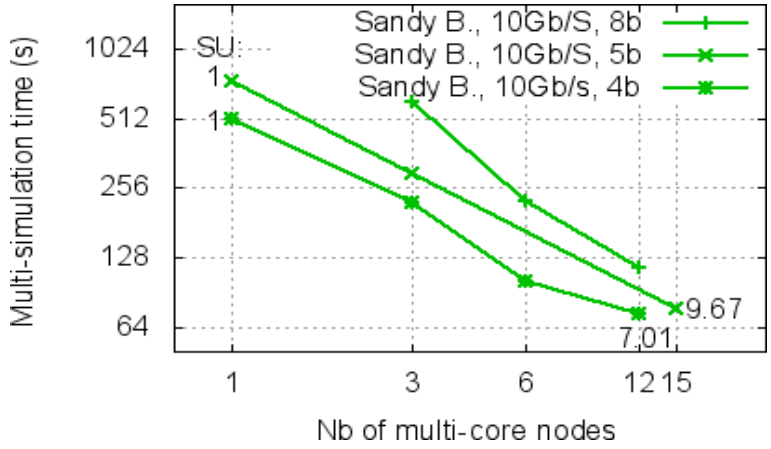

Figure 14: *cl3* benchmark time with 4, 5 and 8 buildings on 10Gb/s cluster

Executions on 6 nodes (for 4 and 8 buildings) should be 2 times longer than executions on 12 nodes. Computing ressources are 2 times less numerous, but communication achievement is also different. The *BC* FMU is connected to 6 local FMUs instead of 3 and achieves 2 times more intra-node communications, and the network switch is solicited on 6 ports instead of 12. Deep investigations would be required to analyse the origin of these abnormally good performances on 6 nodes, but they point out our straightforward deployment (replicating the 1-building best one) is not optimal. A global deployment of b buildings (i.e. $1+10\times b$ FMUs) on n multi-core nodes could achieve better performances, but is not currently available.

7 Conclusion and future work

7.1 Summary of DACCOSIM scaling experiments

Our FMI based multi-simulation environment (DACCOSIM) is multithreaded and distributed, and allows the user to distribute FMU graphs on clusters of multi-core nodes. According to our distribution strategies, our experiments have achieved significant speedup and satisfying size up on an Ethernet 10Gb/s cluster of hexa-cores, running up to 81 FMUs and using up to 16 nodes to simulate heat transfer inside buildings. Complementary large benchmarks replicating an elementary pattern distribution have shown DACCOSIM can *scale* (i.e. efficiently running larger problems on more computing nodes).

7.2 Technical and scientific future work

In order to improve DACCOSIM performances on distributed architectures, our next works will consist in improving both communications and FMU graph distribution:

- First implementation of DACCOSIM communications does not track maximal performances. They will be technically improved: (1) exchanging more compact messages, (2) using high performance MPI communication library instead of ZeroMQ middleware, and (3) exploiting low latency and high bandwidth Infiniband cluster network by relying on MPI.
- Current FMU graph distribution is under the responsibility of the multi-simulation developer, following a methodology deduced from our first experiments. We aim to design a distribution algorithm to point out the ideal number of nodes and the associated distribution of a FMU graph, and also to compute the optimal distribution when the number of nodes is constrained by the cluster availability.
- Some others benchmarks have to be designed and run with variable time steps, in order to measure performances on distributed systems when supporting rollbacks with *ordered* and *overlapped* orchestration modes. Automatic switching between the two orchestration modes, function of the frequency of the encountered rollbacks, will be investigated to attempt to optimize performances.

Finally, the use case considered in this paper was based on multi-floor buildings with equation-based models only. In the future, EDF R&D intends to run hybrid multi-simulations involving cyber and physical models mixing piecewise constant signals and continuous time signals. As FMI-CS has been designed only for continuous time signals, some extensions to the standard FMI are required and some proposals are on the table (see [4] from UC Berkeley). EDF also leads a French working group on this topic with partners (CentraleSupélec, Dassault Systèmes

and CEA List). From the point of view of the optimal distribution of a multi-simulation on clusters, the control FMUs will certainly be very light and ought to be handled by the deployment algorithm without disturbing the distribution of the heavily calculative FMUs.

Acknowledgment

Authors want to thank Region Lorraine for its constant support to their research project, and Robin Armant and Gabriel Pichot (CentraleSupelec students) for their design of some multi-simulation graphs and their investigations about DACCOSIM communications.

References

[1] *Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems*, volume Proceedings of the 10th International Modelica Conference. Modelica Association and Linkoping University Electronic Press, March 2014.

[2] D. Broman, C. X. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. Determinate composition of FMUs for co-simulation. In *International Conference on Embedded Software (EMSOFT)*, 2013.

[3] C.-C. Chang, G. Czajkowski, T. von Eicken, and C. Kesselman. Evaluating the Performance Limitations of MPMD Communication. In *Supercomputing, ACM/IEEE Conference*, 1997.

[4] Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks, and Edward A. Lee. FIDE - An FMI Integrated Design Environment, October 2015. Poster presented at the Eleventh Biennial Ptolemy Miniconference, Berkeley.

[5] Mohammad I. Daoud and Nawwaf Kharma. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 68(4), 2008.

[6] Jorg Dummler, Thomas Rauber, and Gudula Runger. Mapping algorithms for multiprocessor tasks on multi-core clusters. In *Parallel Processing, 2008. ICPP'08. 37th International Conference on*, Sept 2008.

[7] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley, 2000.

[8] V. Galtier, S. Vialle, C. Dad, J.-Ph. Tavella, J.-Ph. Lam-Yee-Mui, and G. Plessis. FMI-Based Distributed Multi-Simulation with DACCOSIM. In

Proceedings of the 2015 Spring Simulation Multiconference (TMS/DEVS'15), 2015.

[9] Hanno Georg, Sven C. Müller, Nils Dorsch, Christian Rehtanz, and Christian Wietfeld. Inspire: Integrated co-simulation of power and ict systems for real-time evaluation. In *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, Oct 2013.

[10] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury. Epochs: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components. *IEEE Transactions on Power Systems*, 21(2), May 2006.

[11] Frederick Kuhl, Richard Weatherly, and Judith Dahmann. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall PTR, 1999.

[12] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[13] Shie-Yuan Wang, Chih-Che Lin, Yan-Shiun Tzeng, Wen-Gao Huang, and Tin-Wei Ho. Exploiting event-level parallelism for parallel network simulation on multicore systems. *IEEE Transactions on Parallel and Distributed Systems*, 23(4), April 2012.